AMOS

NEWSLETTER ISSUE 4 - AMOS User Club

Hello and welcome to another issue of the AMOS Newsletter. In this issue we'll be doing the first full and accurate review (not a preview) of the **AMOS Compiler**, as well as a review of the new **CTEXT** extension. We also have our regular features of learners, beginners and expert programming articles and the latest news of AMOS P.D and Licenseware. Not to mention 10-liner games and a list of what to get if you are trying to develop AMOS software seriously.

☐ Out Now !

The Second new add on for AMOS has now been released, the **AMOS Compiler**. This program takes your AMOS programs and turns them into high speed machine code, both speeding them up and saving memory. The Compiler should be in the shops as you read this, or you can buy it from Europress direct. It costs **£29.99**. Read the review in this issue.

☐ Tome Goodies

As well as the compiler upgrade for **TOME**, a Goodies Disk of extra TOME programs will be released soon. The disk will include a 3D Dungeon Display (like Captive and Dungeon Master) routine, Magic Forest II (A scrolling platform game) and an Isometric 3D game (like Airball, Zaxxon etc). The disk will also include extra utilities. This disk will only be available to Registered TOME users when it is released on the 1st of August, and will cost **£5**. The Compiler Upgrade for TOME is nearly finished, and will be sent out free to everyone who registered their AMOS TOME. It includes a new version of the extension, to work with AMOS 1.3 and the Compiler, which includes 5 new commands (bringing the total to 32), as well as a new version of the TOME map editor, which includes a map overview mode, so that you can look at the whole map on one screen (down to 1 pixel per tile) and move around it this way. The manual has also been extended for the new version. All copies of AMOS TOME sold after the upgrade is finished will also include the upgrade disk (making it a 2 disk package), as well as a Backup Label to put on your backup of AMOS TOME .

AMOS TOME is available direct from

Inside this issue:

Bug Report..............................2.
Absolute Beginners......................3.
Scrolling Text..........................5
Learners (Multiple Bullets).............6.
Mini Maths 4 (Vectors)..................7.
AMOS Extensions ....8.
Conversion ....9.
CText (Review)..........................13.
Compiler (Review).......................14.
Moving Things Behind Things...15.
Essential AMOS Developers Guide..........16.
P.D News................................17.
Text Input Routine......................18.
10 Liners: Ping & Brickout......19.
Mini Maths 5 (Decimal Places).....20.

the AMOS Club for **£19.99** for members (**£24.99** for non-members) with a postal cost of **£5** for overseas orders. If you are in Australia, **Pactronics Pty** are also selling AMOS TOME, so you don't have to order it from the U.K.

❑ Selling AMOS

A new agreement reached between the AMOS Club and Europress Software (Mandarin Software's new name), now allows AMOS programmers to hide the fact that their programs were written in AMOS, until 2 months after they have been released. This means that you can send your game off to a software company without saying it was written in AMOS, so that they will look at it without prejudice. 2 months after the program is released (or up to that point), you should write a letter to Europress Software (address at the start of the AMOS Manual) to tell them about the program. If you want, you can still stick to plastering the AMOS logo all over your program though.

❑ Resubs

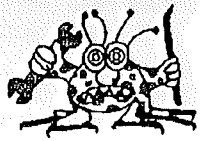
Don't worry, you don't have to resubscribe to the AMOS Newsletter until issue 6 is released. This means that you still have a couple of information packed newsletters to go !

BUG REPORT

If you are using procedures to handle trings and pass them back to the main program, don't ! Because of the way Francois' string system and procedures work, procedures can scramble strings, and even cause system crashes. If your routines need to pass parameters back to the main program, you should use subroutines.

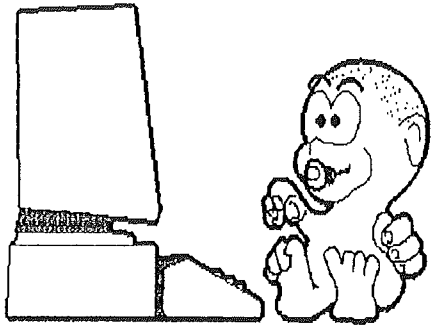
The problem with the speech routines adding the end of a previous message is being worked on. **Jeff Tullin** has managed to come up with a good temporary measure, although it leaves gaps between sentences. Simply make a string up of Underline, Left bracket, Right Bracket and Full stop i.e **K$="_ ( ) . "** and repeat this so that you have a string that is at least as long as the longest string you want to **SAY**. The Speech buffer is based upon the number of phonemes in the string, and the _(). string will force the narrator to create a blank phoneme. Whenever you want to **SAY** a string, set the talking speed to as high as it will go with **Set Talk ,,400** then **Say K$** use the set talk command to set the talking speed back to the speed you want, then say your string. The **Say K$** clears the phonetics buffer, so that your string will not have the end of the previous one tacked onto the end. The following routines should do the job

```
REM Put this at the start of your
 program:
K$="" : L=250
repeat
K$= K$ + "_()."
until len(K$) > L
.....
REM this bit goes at the end of your
 program
Procedure TALK[T$,S]
shared  K$
Say "" : rem null string
Set Talk , , 400
Say K$
Set Talk , , S
Say T$
Endproc
```



Whenever you want to use speech, do the following **TALK[string$,speed]** Both Jeff and myself are working on ways of making this faster (Jeff is working on SAYing single words at a time, while I am trying to see if it is possible to clear the narrator buffer via machine code). Watch this space (Well actually, watch in the next newsletter as this particular space isn't likely to do much).

Absolute Beginners



Principles of Animation...

Before using any of the commands in AMOS that handle objects moving around or animating, you need to know how animation and movement work. This is why many of you are getting stuck trying to use AMAL routines, which most of the magazine articles seem to have gone for. AMAL is only usefull for basic animation and movement of objects or screens. It isn't much use for anything that needs to be controlled, or follow complex patterns. AMAL doesn't speed up when compiled, and is not a very good beginners language, as mistakes are difficult to find and correct. While you get to grips with programming, stick with AMOS commands only.

First steps.

Animating an object is done by constantly presenting different images of the object to the human eye, fooling it into believing it is seeing a moving object. On the computer, there are several ways of doing this. You can display a series of pre-drawn screens (a lot of demos do this), change a single screen by redrawing the graphics, or use colour cycling to create an optical illusion (not much use except for effects). An example of redrawing an object on screen follows. The block will move from left to right on the screen. For the moment, we will ignore problems such as flicker and jerkiness, the intention is to learn how to make things move. You can learn how to smooth them out later. The X co-ordinate of the block is stored in the variable X (logically), and to make the block move a pixel to the right we have to:

1) Erase the block at its old co-ordinate

2) Add 1 to X

3) Re-draw the block at its new co-ordinate

In its most primitive form, it works like this:

```
(1) Curs off : Flash off : Cls 0
(2) X = 1
(3) Repeat
(4) Ink 0 , 0
(5) Bar X , 100 to X + 16 , 116
(6) X = X + 1
(7) Ink 1 , 1
(8) Bar X , 100 to X + 16 , 116
(9) Until X = 300
(10) End
```

Remember, when typing the program in, don't include the line number in brackets.

The small block will flicker its way across the screen. You have taken your first step into movement ! Imagine how difficult it would be to write a typical game, where you would have to re-draw every object each frame (a frame is the time taken for the game to do one loop), even the most simple game would be a nightmare ! Amazingly, most of the original computer games were written using this method. Fortunately, the Amiga (and AMOS of course) has several functions that make life easier for the game writer.

The Amiga can shift chunks of graphics around in memory and on the screen, and at reasonably high speed. In AMOS this is achieved with the **SCREEN COPY** command, which will copy a chunk of graphics from one screen area to another. The next program is basically the same as the first one, except that it uses screen copy to copy the object from a spare screen to do the redraw.

```
(1) Curs off : Flash off : Cls 0
(2) Screen Open 1 , 320 , 200 , 16 ,
 lowres : Curs off : Flash off : cls 0
(3) Ink 1 , 1 : Bar 0 , 0 to 16 , 16
(4) Ink 4 , 4 : draw 0 , 0 to 16 , 16
(5) Screen hide 1 : Screen 0
(6) Rem the above just drew the
 object on a hidden screen
(7) X = 1
(8) Repeat
(9) Ink 0, 0 : Bar X, 100 to X + 15 , 115
(10) X = X + 1
(11) Screen Copy 1, 0, 0, 16, 16 to 0 , X
 , 100
(12) Until X > 300
(13)  End
```

As you can see, whatever is drawn onto screen 1 at the start of the program is moved across the screen (faster than drawing it every frame). One experiment for you to try, is to remove line 9 (the Erase the Object line). When you run the program, you will notice that the object leaves a trail behind it. Again, back in the old days, games were written that didn't use an erase section, the objects were drawn with a black border around them, so that when the object moved, the black border was left as a trail, thus eliminating the need to erase the object. Unfortunately, you couldn't move the object over a background, as it would erase the background !

Going another step up the movement ladder, we come to Bobs and Sprites. The differences between these were explained in issue 1, but basically, Bobs don't present too much diffulty, while Sprites are ****** awkward ! As you can guess, I would recommend using Bobs for most jobs. The following program shows you how to move a bob (loaded from the AMOS Data disk) across the screen.

```
(1) Curs off : Flash Off : Cls 0
(2) Load "AMOS_DATA:
 MAGIC_FOREST / MFSPRITES . ABK"
(3) X = 1
(4) Repeat
(5) Bob 1 , X , 100 , 1
(6) X = X + 1
(7) Until X = 300
(8) End
```

On running this, you can see why most games use bobs rather than redrawing the graphics ! I'll now explain the Bob command. The bob command has 4 parameters. These are:

Bob Number : This is the invisible object that carries the image assigned to it. A good, although rather silly anology is that of invisible demons holding up pictures. When you move the demon, the picture looks like it is moving. When you refer to the Bob number, you are refering to the 'demon'

X Co-ordinate : This is how far across the screen the bob will be positioned.

Y Co-ordinate : This is how far down the screen the bob will be positioned.

Image Number : This refers to the images stored in your sprite bank, and is the Picture number held by the 'demon'

As explained previously, you can have as many bobs on screen as you want, as long as they all use a different bob number. You can use the same image number as many times as you want however. The Hot Spot of the bob determines which point the image is being held by the 'demon'. So that if the hot spot was at the top left of the image, then that bob displayed at 70,80 on the screen would have its top left point at that co-ordinate. Control over a bob during a game, is simply done by changing the bob's X & Y co-ordinates according to joystick movements. For instance

```
(1) Curs Off : Flash Off : Cls 0
(2) Load "AMOS_DATA :
 MAGIC_FOREST / MFSPRITES .
 ABK"
(3) X=160
(4) Repeat
(5) Bob 1 , X , 100 , 1
(6) If Jleft(1) Then X = X - 1
(7) If Jright (1) Then X = X + 1
(8) Until Fire(1)
(9) End
```

This will allow you to control the bob left & right. See if you can make it so that you can control up & down movement. Next issue, I will cover animation.

THE AMOS GAMES MAKER'S MANUAL

This book by Stephen Hill should be ready for release sometime in October. It will include sections on various types of games writing, including **Adventure games, Bat & Ball games, arcade games** and **Map based games**. It will also include sections on the **Compiler & TOME.**

The book will assume that the reader can program in BASIC, but hasn't programmed in AMOS before, and Stephen is taking great care to make sure that the book has the right level of detail.

We'll have a review of the Games Maker's Manual as soon as it is about to be released.

Once finished, it will be available from most bookshops, and will be published by **Sigma Press**.

The following routine contains the basics for scrolling a text message across a screen. This sort of stuff is a bit old hat for demos, but you never know when it might come in useful.

The basic idea is:

1) Place character on the right of the screen

2) Scroll text band across by 1 pixel (Done by screen copying, as it is smoother than Def Scroll/Scroll)

3) continue doing 2 until you have scrolled the width of 1 character, and then go back to (1) to display the next character.

```
(1) Rem  Simple  text  scroller
(2) T$="  Hello  this  is  a  scrolly
 message . . .  "
(3) Screen Open 1 , 320 , 20 , 16 ,
 Lowres : Curs Off : Flash Off
(4) Hide On  :  Cls  0  :  Screen
 Hide 1  :  Ink 1 , 0 , 1
(5) Screen 0 :  Curs Off  :
 Flash Off  :  Cls 0 :  Wait Vbl :
 Double Buffer
(6) P = 1 : X = 0 : Autoback 0 : S =
 2  :  Rem  s is the scroll speed  ( 1 ,
 2 , 4 or 8 )
(7) Do
(8) Screen Copy 1 , 0 , 0 , 312 , 9 To
 Logic ( 0 ) , 0 , 100 : Screen Swap
 : Wait Vbl
(9) Screen 1 :  Screen Copy 1 , S , 0 ,
 320 , 9 To 1 , 0 , 0
(10) X = X + S : Rem this does the
 scroll If X > = 8
(11) X = 0 : Text 312 ,  7 ,  Mid$( T$ ,
 P , 1 )
(12)  Inc  P
(13) If P > Len( T$ ) : P = 1
(14)  End If
(15)  End If
(16)  Loop
```

Learners

This issue, you'll be learning how to use a fire routine for your games, that can use more than one bullet at once. This of course, is quite usefull in shoot-em-ups, and is one of the things that I get asked about on the helpline a lot.

First of all, the theory (don't yawn at the back). To use a single bullet, you should have its co-ordinates stored in two variables, **BX** & **BY** for instance. The bullet is then moved by changing these co-ordinates. For a multiple bullet routine, you need to use an array for each of these variables, such as **BX(n)** and **BY(n)**. Your program will need to remember how many bullets are in flight at any time, so that it knows how many to plot, so we will use the variable **NB** for this. The bullet routine should count through the active bullets, and plot them on the screen (with the BOB command). The bullet should then be moved. When this is done for all of the bullets, you should do a check to see if any of the bullets are 'dead', either off the screen or blown up (when hitting an alien for instance). If a bullet is dead, all the variables used for it should be swapped with the last bullet in the list, and the number of bullets decreased. For instance, we have 4 bullets flying, and bullet 2 is dead.

Bullet 1 X=50,Y=23
Bullet 2 X=50,Y=-1 : DEAD
Bullet 3 X=28,Y=40
Bullet 4 X=93,Y=80 : Last Bullet
Swap Bullets 2 & 4
Bullet 1 X=50,Y=23
Bullet 2 X=93,Y=80
Bullet 3 X=28,Y=40
Bullet 4 X=50,Y=-1 : Dead

and reduce the number of bullets to 3.

The end result is that all the other bullets are still in the list, but the dead one has been removed. This method is similar to the bubble sort (issue 2). To fire a bullet using this method, you simply set all the co-ordinates for element **NB** to the start co-ordinates of the bullet, and increase **NB**. It is always a good idea to make sure that **NB** isn't too high though.

☐ The Roman Candle.

This small program demonstrates how to handle multiple bullets in your programs. Each bullet has 4 variables used in it. **BX,BY** (co-ordinates) **BDY** (Y vector) and **BS** (Sprite Number). The Firework is controlled by the mouse, and pressing the left mouse button will fire a 'bullet' (in this case a star). Pressing the right mouse button will stop the program. As usual, the line numbers are for reference only, so don't type them in.

```
(1) Rem The Roman Candle, by Aaron
 Fothergill
(2) Rem demonstration of firing multiple
 "Bullets"
(3) Rem with gravity on the bullets
(4) Dim BX( 10 ) , BY( 10 ) , BDY( 10 )
 , BS( 10 )
(5) Rem create some bobs
(6) Curs Off : Flash Off : Cls 0 :
 Paper 0 : Pen 1 : Ink 2 , 2
(7) Bar 0 , 0 To 7 , 15 : Hide On
(8) Get Bob 1 , 0 , 0 To 16 , 16 : Hot
 Spot 1 , 4 , 0
(9) For A=0 To 9 : Cls 0 : Locate 0 ,
 0 : Pen A + 2 : Print " * "
(10) Get Bob 2 + A , 0 , 0 To 16 , 8 :
 Hot Spot 2 + A , 4 , 4 : Next A
(11) X = 160 : Y = 180  : Rem co-
 ordinates of firework
(12) Cls 0 : Double Buffer : Colour 2 ,
 $FFF : Colour 1 , $FF0
(13) Fade 1 , $0 , $FFF , $F00 , $F0 ,
 $F , $FF0 , $FF , $F0F , $F80 , $8F0
 , $F8 ,  $8F , $80F , $F08
(14) While Mouse Key < 2
(15) Bob 1 , X , Y , 1
(16) If NB > 0 Then Gosub BULLETS
(17) GTG = 1 - GTG : X = X Screen (
 X Mouse )
(18) If Mouse Key = 1 Then  Gosub
 FREBULLET
(19) Wend : End
(20) BULLETS:
(21) For A = 0 To NB - 1
(22) Bob 2 + A , BX( A ) , BY( A ) ,
 2 + BS( A )
(23) BX( A ) = BX( A ) + 1 : BY( A )
 = BY( A ) + BDY( A )
(24) If GTG = 0 Then BDY( A ) = Min(
 8 , BDY( A ) + 1 )
(25) Next A
(26) NB2 = NB
(27) For A = NB - 1 To 0 Step -1
(28) If BY( A ) > 199 or BX( A ) < 0
 or BX( A ) > 319
(29) Bob Off 1 + NB2
(30) Swap BX( A ) , BX( NB2 - 1 ) :
 Swap BY( A ) , BY( NB2 - 1 )
(31) Swap BDY( A ) , BDY( NB2 - 1 ) :
 Swap BS( A ) , BS( NB2 - 1 )
(32) Dec  NB2
(33) End If
(34) Next A
(35) NB = NB2
(36) Return
(37) FREBULLET:
(38) If NB < 10
(39) BX( NB ) = X : BY( NB ) = Y
(40) BS( NB ) = Rnd( 9 )
(41) BDY( NB ) = Rnd( 2 ) - 12
(42) Inc NB
(43) End If
(44) Return
```

Save it before you run it !

Those of you who looked at last issues mini maths programs, will notice the use of **GTG** as a toggle variable, so that **BDY** is only added to **BY** every other frame.

☐ Mini Maths 4

Using Vectors

Vectors are fractional changes in a value in a particular direction. In terms of objects, a vector is usually a speed of movement in one direction.

Vectors are commonly used in games, as a way of showing the direction of movement of bobs etc. (Both the Brick-out and Ping 10-liners use vectors).

If, for instance, you wanted a ball to move in various directions, you could use 2 variables as the X & Y Vectors for the movement (one storing the amount to move in an X direction and one storing the Y direction movement per frame). These could be called DX & DY (The D being from Delta, a mathematical term which normally represents an infinitely small fraction of a value).

The following program shows you how vectors affect an object. It allows you to enter the X & Y movement vectors and see how the block moves. For each of X & Y, try values of -8 to 8 (including fractions).

```
(1) Input "X Vector " , DX#
(2) Input "Y Vector " , DY#
(3) Curs off : flash off : cls 0 ink 1 , 1
 : Bar 0 , 0 to 3 , 3
(4) Get Bob 1 , 0 , 0 to 4 , 4
(5) X# = 160 : Y# = 100 : rem set to
 centre of screen
(6) While X# > 0 and X# < 320 and
 Y# > 0 and Y# < 200
(7) Bob 1 , X# , Y# , 1
(8) Wait Vbl
(9) X# = X# + DX#
(10) Y# = Y# + DY#
(11) Wend : rem exits when off screen
(12)  End
```

AMOS Extensions

Thanks to a bit of forward thinking by Francois, AMOS can be extended by adding extra commands to it. These extensions can be written in machine code (or C if you are lazy and don't mind the massive speed and memory wastage).

Each extension is installed in into AMOS with its own number and can add whatever new commands you want to AMOS. The only drawbacks are that you can't use any commands already reserved in AMOS, you need a different version of the program for V1.3 (the Compiler) version of AMOS, and you need to make sure you are using an extension number that isn't currently in use (You can't use two extensions with the same number).

Anyone writing extensions must contact us here at the AMOS Club to get an official extension number for their new extension before releasing it. Don't bother contacting Mandarin/Europress software, as they are not keeping a list.

Unless an extension is cleared by the AMOS Club, it will not be counted as an official AMOS extension, and will not be supported.

Currently, the first 8 extensions are in use. These are:

1-Music (Provided with AMOS)

2-Picture Compactor (Provided with AMOS)

3-Requestor Extension (Provided with AMOS V 1 . 2 upwards)

4-reserved for AMOS 3D (Available sometime this year, maybe !)

5-AMOS Compiler (Available now)

6-Serial Extension (Provided with AMOS V 1 . 23 update and AMOS V 1 . 3)

7-AMOS TOME (Available from The AMOS Club since March)

8-CText Extension (Available now on Licenseware for AMOS Club members only)

We will be keeping a list of the extensions here, and you should contact us when your extension is working. If you are using an extension number that has already been reserved, you can change your extension number in your program, and convert all the programs that use it, by saving them as **.ASC** programs first, change the extension number and then merging the programs back with the changed extension. This also solves the problem caused by the extension system in versions of AMOS before 1.3, where changing the number of commands in an earlier extension (or adding or removing it) cause all the commands from later extensions to shift.

Strange but True !

Something Gareth Lancaster and I noticed when Gareth was writing Sandra's catalogue disk, was that if you screen display a screen at Y>309 (in low rez), then any other screens that are actually in view, get moved over to the left, possibly by the copper chip being confused, or just a wierd glitch in AMOS !

Digitizing Service

Technical Fred are now able to print T-Shirt transfers (in colour) as well as digitising still pictures and audio.

Phone them on **061 703 7842** for details.

Conversion

AMOS is actually a very extended and improved version of BASIC (Beginners All Symbolic Instruction Code), which is a common language between just about all home computers except for the unfortunate Jupiter Ace (a 1K RAM machine with a version of the FORTH language, which was a good idea, but it didn't catch on).

The only problem is that every computer has used a slightly different version of BASIC, especially when its version includes commands not covered by the original idea of BASIC, and some versions don't have commands that are in others.

This of course, is a great pity if you want to try out an old listing you (or someone else) did on another computer, or with a different dialect of Basic.

However, with a bit of know how, you can usually translate most (if not all) of the program into AMOS. The trick is in knowing both AMOS and the version of BASIC that you want to convert.

The easiest Basic to convert into AMOS is **STOS** on the ST. This Basic is the "Daddy" of AMOS and most of its commands have direct AMOS equivalents. Plenty of AMOS programmers started on STOS on the ST and then bought Amigas when AMOS came out.

First of all, you should get the program transfered from the original machine or listing, into AMOS. If you have a transfer program that will transfer (and convert) ASCII files between the original machine and the Amiga, then this is the easy method. Otherwise, you will have to type it in ! If you are converting over from STOS, save the program as an ASCII file, port it over with Dos-2-Dos, CrossDos or a coms program and then Merge it into AMOS. You will have to watch out for long lines (as AMOS won't accept lines over about 215 characters long).

Once you have your program in AMOS, you then have to go through it trying to spot all the different commands. For instance, the **MODE** command is used in STOS to change between screen modes (STOS can only display one screen at once). Mode 0 is lowres, Mode 1 is medium and Mode 2 is hires. When used as a command, the Mode command should be replaced with a **SCREEN OPEN** command in AMOS. Mode 0 would be **SCREEN OPEN scrn, 320, 200, 16, lowres** and Mode 1 would be **SCREEN OPEN scrn, 640, 200, 4, hires**. You don't get MODE 2 commands in STOS (As the hires monitor is automatically detected), but you can set the screen to sort of emulate ST hires by **SCREEN OPEN scrn, 640, 400, 2, hires+laced** which won't give you the absolutely "one of the reasons I still use my ST" clear, not flickering and totally readable screen, but it will do. If you see MODE used as a function, it is being used to check to see what mode is in use (0=lowres, 1=medium, 2=hires), and most of the time, you needn't worry about this, just do **MDE=whatever** at the start of your program and change all function occurences of MODE to MDE (to avoid clashing with the MOD function).

If a program uses pokes, peeks, traps, syscalls etc that refer to machine specific memory locations it is a good time to consider giving up, unless you know what the routine or memory location is supposed to do so that you can emulate it in AMOS. If you have all the reference books for the computer you are converting from, you could look up the memory location and work out what it does, then decide on an AMOS equivalent. A lot of the time though, the call will be to the original programmer's own bit of machine code which you might not have any details on. Although with a bit of detective work, you might be able to find out what it does, and how you can emulate it.

Sometimes (very rarely), you might come across a command that AMOS doesn't have any equivalent to. If it is a straight command that doesn't return anything, you could write a procedure to emulate it, or you could use a subroutine to emulate a function. Again, this is a case of knowing what the command is supposed to do in the first place ! The only command known to be just about imposible to do in AMOS (Take the hint Francois !) is the erase an array command for delete arrays. To work this in AMOS, you would have to use memory banks or strings for all your array work, so that you can delete them.

Another point to watch out for when converting programs to AMOS, is that most basics (Except for the Apple][‘s Integer Basic, and STOS) use Floating Point variables as standard, wheras AMOS uses integer variables as standard. So **A=10** in Commodore 64 basic, would be the same as **A#=10** in AMOS. This one catches a lot of people out !

Differing Screen sizes can cause problems, but this can be got around by using co-ordinate multipliers or factors. For instance, you could take the width of your AMOS screen, and divide it by the width of the original screen and multiply all X screen co-ordinates by this value. e.g Apple][screen width was 280 pixels (I know it's a wierd number, but the Apple][used 40 bytes to store a screen line, with 7 bits used in each byte), so you would do: **WM#=320/280.0** and then multiply all X co-ordinates by **WM#**. Another way to work it (especially if the sizes aren't much different) is to just work the program in a smaller screen. Using the **CLIP x1,y1 to x2,y2** command is useful for this. Again, using the Apple][as an example, you could do **CLIP 0,0 to 280,192** to make all your graphics go into an Apple][sized window. You could also use the **SCREEN DISPLAY** command to make sure that this screen is in the centre.

Colours can cause problems, you can either re-work the graphics and their colours to something better, using AMOS to the full, or you have to work out equivalent Amiga colour values for the colours used. If it was on an ST, simply multiply all Red, Green and Blue RGB values by 2 and add 1. So **$435** would become **$97B** (4x2+1,3x2+1,5x2+1), this doesn't apply if the ST program is using STE colour settings, which are the same as the Amiga's. This should give you a basic idea of what to do, there are more things (such as the STOS screen display system) which can give you problems, but we'll cover these in another article.

One more point to watch out for is differing conventions between dialects, especially where non standard commands are concerned.

For instance BBC Basic's Procedures are different from AMOS procedures (as they don't use local variables), and BBC basic's integer variables stay in memory until changed, even when you run the program more than once ! (To emulate this in AMOS, poke, doke or loke the value into a DATA bank and peek, deek or leek it back again when the program runs). Another strange BBC quirk is that the screen co-ordinates start with 0,0 at the bottom left (like graph paper and the strange Deluxe Paint III) . Just about every other language has 0,0 as top left. This usually requires careful conversion o screen co-ordinates (That or upside down graphics). You may also find minor name differences between the same command in different basics (watch out for American Spelling !).

amos TOME

SHADOW SOFTWARE

Now being used by **ALL the major AMOS Programmers**, **AMOS TOME** has just been upgraded to **V3.1** to be compatible with the AMOS Compiler and AMOS V1.3. It now has **34 commands**, and the editor is **still the most powerful map editor available on ANY computer.**

TOME has already been used to write 3 commercial games, and many more are being developed. The update disk is being sent out free to all registered AMOS TOME users once finished, and will be included in all further AMOS TOME packages.

On August 1st, a **TOME Goodies disk** will be released (available only to registered TOME users) including a **high speed scrolling 3D isometric game, 3D dungeon game** and **Magic Forest II**, a horizontally scrolling platform game, as well as an extra manual for AMOS TOME programmers wanting to use these routines in their own games.

AMOS TOME costs **£19.99** for AMOS Club members (**£24.99** for non members) and is only available direct from Shadow Software in the U.K (or from Pactronics Pty Ltd in Australia)

When ordering AMOS TOME, please make cheques payable to Shadow Software and add **£5** postage & packing if ordering from outside the U.K.

Send your order to:

Shadow Software, 1 Lower Moor, Whiddon Valley, Barnstaple, N.Devon. EX32 8NW.

All prices are inclusive of V.A.T at 17.5%

(So you don't have to add any more !)

....

Please send me AMOS TOME (V3.1).....................

Name ....

Address...

....

....

Postcode..**Membership Number**.........................

I enclose a Cheque/Postal Order for **£19.99** (members only), **£24.99** (Non-Members) + **£5** P&P if ordering from Overseas. (P&P is free in the U.K)

AMOS CTEXT

A brand new AMOS Extension, 5 new commands, gives you easy to use icon based text display. Ctext can do:

Fonts of up to 64 colours,
Any size of character,
Easily edited fonts (Can be edited in SpriteX)
Masked fonts,
Proportional text,
Kerning (In the 1.3 version)

Ctext is supplied in AMOS 1.23, 1.3 and ASCII versions with a font setter program, demonstrations and example P.D fonts.

It is **only available to AMOS Club members**, and costs £3.50 U.K, £3.75 Europe and £4 Overseas. **Available only from Deja Vu Software.**

……

Order Form **Send To: Deja Vu Software,**
Please send me: **25 Park Road, Wigan, WN6 7AA**

AMOS SPRITEX (ACLUB 1) ☐£3.50 (£3.75 Europe, £4 R.O.W)

AMOS CTEXT (ACLUB 2) ☐ £3.50 (£3.75 Europe, £4 R.O.W)

The Latest Deja Vu AMOS P.D & Licensed Software Catalogue (Free)

Name...
Address...
....
....
Postcode..............................**Membership Number**..............................

I enclose a cheque/Postal Order for **£**.......... (please make payable to AMOS P.D Library)

……

AMOS SpriteX

The enhanced version of the AMOS Sprite Editor, the latest version now includes:

Auto scrunch (automatically removes waste area from sprites)
Can edit Sprites, Bobs and Icons directly (including the correct palettes)
Includes built in bob grabber and animator.
Automatic NTSC detection.
Now supplied as AMOS 1.23 and 1.3 versions

SpriteX costs **£3.50** (£3.75 Europe, £4 Overseas) and is **only available to AMOS Club members from Deja Vu Software.**

CTExT

Another new AMOS Extension !

If all the people who have had problems using disk based fonts in AMOS were to put their hands up, the Earth's centre of gravity would actually raise by about a foot.

In order to preserve the current centre of gravity and protect the enviroment from the disasters this would cause, the **CTEXT** extension has been developed. Currently being tested by several AMOS developers, and about to be released.

CTEXT (Colour Text) is a new extension to eliminate the problems involved in using different fonts in AMOS. **CTEXT** is installed as extension number 8 (see the article on extensions) and gives you 2 new commands, a new function and 2 new reserved variables (So far).

These are:

Font Size x,y Lets you set the spacing for your font. Using X or Y values of zero allows you to use proportional fonts (different width & baseline for each character).

Ctext x,y,string This command works exactly the same as the Text command, except that it uses an icon bank for its font. This means it is easier to load in a font. It doesn't fragment memory and you can use colour fonts (up to 64 colours). You can also edit your fonts in SpriteX (which can edit icons as well as Bob/Sprite images).

The new function is **=Plen(string)** which returns the **PIXEL** length of a string. This is so that you can find out exactly how long a piece of text will be on the screen.

There are also 2 reserved variables. These are **Font Data** and **Font Base**. These return the addresses of the Font's data table and the base address of CText's variable table.

CText also supports **Kerning**, so that you can move individual characters closer or further away from each other. By adding a **Chr$(27)** (The ESC character) into your string, along with a character to determine the amount of kern, you can control your text output exactly. The Plen function automatically takes account of any kerning when working out the length of the string. For example you could use:

```
T$ = "T" + chr$ ( 27 ) + chr$( 45) +
 "ake this for example"
Ctext 10 , 20 , T$
```

The **chr$(45)** after the **chr$(27)** determines the amount of kern. It is based around character number 48 (the zero) which will give no kern, each character lower kerns left by one pixel, and each higher kerns right.

Ctext will be released on Licenseware (limited to AMOS Club Members, like SpriteX) and will include example fonts and a Font Setter program. Used with SpriteX, you have a full font editing kit.

So far Ctext is already being used by several AMOS developers, including in the CDTV version of the Fun School series. Ctext is available now and includes V1.23 and V1.3 (Compiler) versions of the extension.

Like SpriteX it is limited to **AMOS club members only**, and costs **£3.50**. (**£3.75** Europe, **£4 R.O.W**) Phone Sandra at the AMOS P.D Library (**0942 495261**) for details of how to get it.

As with AMOS TOME, several programs are being developed to support CTEXT, such as extra font disks (for the terminally lazy) and new editing facilities. These should be available on P.D

AMOS COMPILER

The AMOS Compiler is the second add on for AMOS (following on from AMOS TOME) and has at last been released after about 6 months delay.

There were several problems during the development of the Compiler, including a complete re-write of the extension system being needed. As one of the developers testing the compiler throughout its development, I am rather disapointed that Mandarin/ Europress released the compiler before it was completely finished (Francois is just finishing off the new master), but at least it is available and mostly works. There are still very small problems with large programs, but it will compile just about everything.

The Compiler costs **£29.99**, which is reasonably cheap for a compiler, no doubt the bucket merchants will be selling it for considerably less quite soon, and look out for good offers at the shows.

It comes with an update disk to update your version of AMOS to V1.3 and comes with several small demos, although most people will be more concerned with how fast it will make their own programs.

The manual is bound the same as the first release of the AMOS manual, i.e fold it flat and it breaks ! AMOS should have been released with a folder based manual so that extra manuals could be added in.

Once the compiler extension is installed as extension 5 (a very painless job), you can load in the Compiler program and try compiling your programs. You can use the Compiler direct from the CLI also, once you get used to it, but I prefer the more friendly enviroment of AMOS.

Compiling is usually a case of Click on the Compile button, choose what program you want to compile, and then choose what you want to call the compiled file. The compiler does the rest. If you want, you can play around with the options, which allow you to do things such as squash the compiled program (saving about 40% of the disk space) and whether or not you want it to open a default screen.

Programs can be compiled as CLI or Workbench run only programs, or as an AMOS Procedure, which can be included in your AMOS programs (it can't be opened of course). If you compile the program as a CLI command, you can even pass parameters from the CLI to your program, so you can add your own commands to a startup sequence for instance !

The Compiler isn't yet perfect however, some programs will compile perfectly, but show strange results when running, usually a couple of commands skipping or functions returning null values. You also have to be more careful with strings in procedures, as memory can quite easily fragment. This only seems to happen on large programs however, and can usually be worked around. This seems to be the result of Mandarin rush releasing the Compiler before Francois finished the final version. Hopefully, Mandarin will be sending out a finished version to everyone who registers their Compiler.

Having worked with the Compiler for a couple of months in development, and for a month with the release version, I am very impressed by its results. Apart from a few problems it can cause with some programs, it works well, and should be part of any developer's arsenal. Go out and buy it.

The AMOS Compiler costs **£29.99** and is available from most computer shops, or direct from: **Europress Direct, Europa House, Adlington Park, Adlington, Macclesfield, SK10 4NP.**

Moving things behind things

O.k so, last issue I forgot ! Here's the article that would have gone in last issue if there had been space.

Apart from using Dual Playfield mode (not a good idea), there is one reasonably workable way of making bobs able to go behind scenery, or other bobs. To use this system, you have to follow a set of rules in how you display your screen.

Fortunately, this system is also usefull if you are trying to use backgrounds that change behind your bobs (if you do this wrong you get 'trails' behind your bobs).

First of all, you should open up a spare screen to store the background on, and hide it. If you were using screen 0 to display, you could open (and hide) screen 1 as your background screen. Then, of course, make sure that your background is on this screen.

You would then use the following sequence as the first part of your main game loop (i.e do this every frame).

```
Bob Clear
Screen Copy 1 to Logic( 0 ) : rem you
  can also just copy a part of the
  screen if your game area is smaller.
screen  0
rem do a Limit Bob x1 , y1 to x2 , y2
  here if you need it.
rem do all your BOB commands here,
  any done outside this bit will go
  funny !
Bob  Draw
rem anything placed on Logic( 0 ) at
  this point (either by Screen copy, Put
  Block or Paste Icon) will be in front
  of the bobs
Screen  Swap
rem a Wait Vbl command can be used
  here if needed.
```

The theory behind this system is that first you clear all the bobs (so the backgrounds don't interfere), then copy the background to the hidden logical screen (anything placed on the logical screen will be seen when you do a screen swap). You then place all your bobs, (put your **SYNCHRO** command in at this point if you have to use AMAL). The **Bob Draw** command then redraws them on the logical screen, at which point you can overlay anything you want to be in the foreground, with **Paste Icon** (for small objects), **Put Block** or **Screen Copying** (if you don't need colour 0 to be transparent).

THE AMOS HELPLINE

Remember that as an AMOS Club member, you can call on the AMOS Helpline if you get stuck.

You'll need your membership number, so have your card handy (write your membership number on all your newsletters !).

☎0271 23544

The best time to phone is between 2pm and 7pm, as this is when I'm most likely to be in, and awake (most of this newsletter was written at 1 a.m in the morning). It's O.k to call outside these times, but I might not be in.

You can also write in with programming queries, but enclose a stamped self addressed envelope, or you won't get a reply.

Between the 15th of August and 1st of September, I will be away, so phone Europress on **0625 859333** for technical help. They aren't very good at it (I bet they tell you to phone me !), but they might be able to help !

The Essential AMOS Developers Guide

There are now several programs out that can be used with AMOS, or were written specifically for it that anyone developing programs in AMOS should have, and there are a few that make life a lot easier, but are optional. Here is the first list of programs, we'll update it as new stuff is released.

A 1 Meg Upgrade and a second drive: Absolutely vital if you are serious about writing any software at all. 1 Meg upgrades are now available for **£20**, and once you can afford it, second drives are down to **£50** (The same price as AMOS was when it was first released !)

The AMOS Compiler: This should be the first add on you buy for AMOS (Once it is released), it speeds up your programs, turns them into proper run only files and even reduces the amount of memory they use. Well worth **£29.99** despite some bugs in early version. Available now, Most Shops

AMOS TOME: This is vital if you are writing Map based games. Even if you have your own display routine, the TOME editor is the most powerful map editor available. **£24.99 (£19.99** to AMOS Club Members), available direct from the AMOS Club (Or Pactronics in Australia).

AMOS 3D: Not very impressive, and overpriced at **£34.99**, wait for any one of the several **P.D 3D extensions** that are being developed.

CText: If you are using custom fonts, or want to use proportional or colour text in your programs, then **CTEXT** is a must. It has been released as Licenseware (available to AMOS Club Members only) and costs **£3.50.** Contact Sandra Sharkey for availability, or Aaron at the AMOS Club for technical details.

Sprite X: This is the upgraded version of the Sprite Editor, which includes the sprite grabber and an animator built in, also several automation features to make sprite editing easier, and the ability to edit icons. Available from the AMOS P.D Library as Licenseware, Sprite X is only available for AMOS Club Members (so include your membership number) and costs **£3.50**, an older version of Sprite X is also included with the AMOS Compiler.

Rainbow Warrior: Not only is this one dead useful, it's also on P.D. Great for boring rainbows in demos (yawn), as well as being incredibly good for doing rainbow game backdrops, which really beef up the graphics of a game. Available from the AMOS P.D Library **£2.50**

An Art program: The choice of art package you use is up to you (or your graphics artist). They range from the very cheap (P.D) **AMOS Paint**, which is good enough for basic work. Up to **Deluxe Paint III** (which is wonderful). Currently Adam (My brother and graphics artist) is jumping up and down waiting for **Deluxe Paint 4**. Another paint package recomended by Sandra is Spectra Paint (APD 134) which being P.D, costs **£2.50**

If you (or your graphics artist) are good at graphics, then you should make sure that you find a paint program that you like, and can work with. It is also important to find a mouse, graphics tablet or even trackerball that you like working with.

A Music Editor: This is basically either **GMC** (APD 1), **M.E.D** (V2.1 or below) or your favourite 4 voice **Soundtracker** program. Our current favourite is **PRO TRACKER**, which can be directly converted to AMOS format by the current Soundtracker2.1 converter program (on the V1.21/1.23 update disk). So far though, I haven't found a really good music editor on the Amiga.

AMOS P.D News

Due to mistakes by 17 bit software and Mandarin (Now Europress) software, there has been some confusion over AMOS Licenseware programs . . .

First of all, AMOS Licenseware programs are commercial software, selected by Sandra Sharkey. They are not P.D disks, and only Sandra decides what goes on Licenseware. Secondly, Licenseware distributers must sell the disks at **£3.50 (£5.50** for 2 disk programs), the programmer receives a royalty from this.

Recently Mandarin decided that none of the programs entered for the Games Writer of the Year competition were good enough, and sent all the disks back to the writers with a letter telling them to send them in to be included as Licenseware.

Basically, if you got one of these letters, ignore it, as **Mandarin have no say in what goes on Licenseware.** If you think your program is good enough for Licenseware then send it in to Sandra. If she agrees, it will go on Licenseware, otherwise, she will ask you if you want to release it as P.D or Shareware.

17 Bit managed to annoy the other Licenseware distributers by selling "Packs" of Licenseware disks at well below their correct price, and also advertising them as P.D. They have been told off, and have published an apology in their latest advert.

If you see any of the libraries advertising AMOS Licenseware below its correct price of **£3.50,** tell Sandra so it can be sorted out. Remember also, that you cannot copy Licenseware, as it is still the copyright of its author. Copying it is illegal, and is worse than copying full price software (as the author of a Licenseware title doesn't get the security of an advance on royalties that a full price author gets.

A new scheme introduced to stop illegal copying of Licenseware disks is Sandra's Licensed Software Labels scheme. All legal AMOS Licenseware disks after the 1st of August will have an Official Deja Vu Licensed Software label on them. If the label is not on the disk, it is an illegal copy, and you should report it to Sandra at the P.D Library, telling her where you got the disk from.

□ Deja Vu Software

This is the new name for Sandra Sharkey's combined Licenseware and the official AMOS P.D library.

There are some new disks about to become available from Deja Vu as Licensed Software: All are **£3.50** U.K, **£3.75** Europe, **£4.00** R.O.W

Derek Pell has perfected the screen dump routine. His routine calls the CLI screen dump command so that you can dump to any printer (including colour) via the Intuition. You can set your printer type with the Preferences tool from workbench. The disk includes a demo and documentation.

Phillip Richmond's Adventure creator is work a look for all adventure fans. It includes a very powerful parser, and can save adventures out as stand alone programs, which can be compiled. It can handle text, graphics and music, and comes with a demo adventure.

The follow up to Dogfight II, **Galleons** has just been released **(LPD3)**. It is an arcade combat game, where you control sailing ships, trying to wipe out the other 3 captains. The game is highly detailed, sails and masts collapse as they are hit, ships can catch fire, and you even get puffs of smoke drifting away as the guns are fired. There are also bonuses to collect and the game even includes animated cartoons in the credits. Great fun for 1-4 players .

Text Input Routine

You might find this routine useful if you need to limit input to a set length, or use your own input routine for some reason (such as for high scores).

```
(1) Rem  Basic  Line  Input  routine
 , returns string in A$
(2)  Rem  uses  TEXT  command  to
 display , so  you  can change fonts
(3) Rem  or  use  the  CTEXT
 extension to displa
(4)   Rem
(5) Curs Off : Flash Off : Cls 0
(6) Locate 10 , 10
(7)  Gosub  GTLINE
(8) Locate 0 , 0
(9) Print A$
(10)  End
(11) GTLINE: Rem returns String in A$
(12) Ink 2 , 0 , 2
(13) L = 20 :  Rem  this  is  the max
 length  of the line ( change it if
 you want)
(14) L$ = " . " : Rem this is the
 Basic line character , change it to
 whatever
(15) Rem  you  like  ( spaces   for
 instance )
(16) A$ = String$ ( L$ , L )
(17) X = X Graphic ( X Curs )
(18) Y = Y Graphic ( Y Curs ) + 8
(19) Text X , Y , A$
(20) P = 0 : OK = 0 : While OK = 0
(21) B$ = "" : While B$ = "" : B$ =
 Inkey$ : Rem key in
(22) Text X + P * 8 , Y , Mid$ ( A$ ,
 P + 1 , 1 )
(23) Draw X + P * 8 , Y - C To X +
 P * 8 + 7 , Y - C : C = ( C + 1
 ) mod 7
(24)  Wait Vbl  :  Wend
(25) Text X + P * 8 , Y , Mid$ ( A$ ,
 P + 1 , 1 )
(26) If ( B$ > = Chr$ ( 32 ) ) and ( B$
 < = Chr$ ( 126 ) )
(27) A$ = Left$ ( Mid$ ( A$ , 1 , P )+
 B$ + Mid$( A$ , P + 1 ) + String$( L$
 , L ) , L )
(28) Text X , Y , A$
(29) Inc P  :  P = P mod L
(30)  End  If
(31) If B$ = Chr$ ( 13 ) :  Rem
 RETURN   key    pressed
(32)  OK = 1
(33)  End  If
(34) If B$ = Chr$ ( 8 ) : Rem back
 space
(35) If P > 0
(36) A$ = Left$ ( Mid$ ( A$ , 1 , P - 1
 ) + Mid$ ( A$ , P + 1 ) + String$ (
 L$ , L ) , L )
(37)  Dec   P
(38) Text X , Y , A$
(39)  End  If
(40)  End  If
(41)  Rem  check  for arrow left or
 right
(42) SC = Scancode
(43) If SC = 79 : Rem left arrow   (
 move cursor left )
(44) P = ( P + L - 1 ) mod L
(45)  End  If
(46) If SC = 78 : Rem right arrow   (
 cursor right )
(47) P = ( P + 1 ) mod L
(48)  End  If
(49) If  SC = 70 : Rem  delete  key
(50) A$ = Left$ ( Mid$ ( A$ , 1 , P ) +
 Mid$ ( A$ , P + 2 ) + String$ ( L$ ,
 L ) , L )
(51) Text X , Y , A$
(52)  End  If
(53)    Wend
(54)   Return
```

To change this routine to use CTEXT, simply change all the TEXT commands to CTEXT and remember to add in the CTEXT font loading routine at the start. One point, don't try using this routine as a procedure, you should never use procedures for anything that requires strings.

10 Liners

2 bat and ball games this issue. Brickout (which was forced out of the last issue by lack of space) and Ping. You will need to design a few sprites for these, but nothing complex is needed, just a bat, ball and a few bricks for Brickout.

In Brickout, you must bounce the ball to destroy the wall of bricks (sounds familiar doesn't it ?), and in Ping, you must beat the computer player. In levels 1-3 of Ping, the game plays normally, but in levels 4,5 & 6 the ball jiggles in flight, making life more difficult.

Sprites required for Brickout:

1-Bat, about 6 pixels high, 32 wide, hotspot in centre

2-ball, 2x2 pixels, hotspot top left

3-8 bricks, 16x8 pixels, various colours, hotspot at top left

Sprites required for Ping:

1-Bat (vertical) 8x32 pixels, hotspot in centre

2-Ball, 2x2 pixels, as brickout.

☐ **Brickout**

(Don't enter the line numbers in brackets !)

```
(1) Dim T ( 19 , 5 ) : Double Buffer :
 Curs Off   :  Reserve  Zone  100 :
 Hide On  :  Cls 0 : LIVES = 3 :
 Get Sprite Palette
(2) LVL = 1 : LVLO = 1 : While LIVES >
 0 : CO = 0 : I nk  14 : Bar 0 ,
 0 To 319 , 199  :   For A =
 0 To LVL - 1  :  For  B = 0 To 19
(3) T ( B , A ) = 1 : Paste Bob B * 16
 , A * 8 + 30 , ( A mod 4 ) +
 3 : Set Zone A * 20 + B + 1 , B * 16
 ,A * 8 + 30 To B * 16 + 15 , A * 8
 + 37 :  Next B  :  Next  A
(4) X = Rnd ( 100 ) : DX = Rnd ( 16
 ) : DY = - 1 : Y = 190 : While Y <
 198  and LVL = LVLO  :  Bob 1 ,
 X Screen ( X Mouse ) , 190 ,
 2 : Bob 2 , X / 16 , Y , 1 : X = X +
 DX : Y = Y + DY
(5) Z = Min ( 100 , Zone ( ( X + DX ) /
 16 , Y ) - ( 101 * ( ( X + DX ) / 16 <
 0 or ( X + DX ) / 16 > 319 ) ) - 1
 ) : If Z > = 0 and ( T ( Z mod 20 ,
 Z / 20 ) or Z = 100 )  Then  Gosub
 ZAP : DX = - DX
(6) Z = Min ( 100 , Zone ( X / 16 , Y +
 DY ) - ( 101 * ( Y + DY < 0 ) ) - 1
 ) : If Z > = 0 and  ( T ( Z mod 20 ,
 Z / 20 ) or Z = 100 ) Then Gosub
 ZAP : DY = - DY
(7) LVL = LVL - ( CO > = LVL * 20
 ) : VX = ( MXO - X Mouse ) : MXO =
 X Mouse : C = Bob Col ( 1 ) : If
 C Then DY = -1 : DX = Max ( -32 ,
 Min ( 32 ,  DX + VX ) )
(8) Wend : LIVES = LIVES + ( LVL =
 LVLO ) :  LVLO = LVL : Wend : End
(9) ZAP: ZX = Z mod 20 : ZY = Z /
 20 : T = T ( ZX , ZY ) : CO = CO +
 T :  SCORE = SCORE + T * ( ZY +
 1 ) * 10 :  T ( ZX , ZY ) = 0 : Ink 14
(10) Bar ZX * 16 , ZY * 8 + 30 To  ZX
 * 16 + 15 , ZY * 8 + 37 : Bell 4 :
 Return
```

☐ Ping

(Enter the line numbers as they are needed)

```
10 Dim D ( 1 )  : Curs Off  :  Flash
 Off  : Cls 0  :  Screen  Open 1 , 320
 , 200 , 16 , Lowres :  Paper 0 :
 Pen 1 : Curs Off : Flash Off :
 Screen   Hide  1  :  Screen  0  :
 Paper 0 : Locate 0  ,  12  :  Centre
 "P I N G" : Locate 0 , 23 :
 Centre "Press A Key" : Wait Key :
 Print
20 LVL = 0 : While LVL < 1 or LVL >
 6 :  Input "Level of difficulty (1-
 6) ?"; LVL :  Wend :  Curs  Off :
 WG = ( LVL - 1 ) /3 : LVL = ( ( LVL
 - 1 ) mod 3 ) + 1
```

```
30 Hide On  :  Cls  0  :  Double
 Buffer : Get Sprite Palette :
 Screen 1 : Autoback 0 : Y1 = 100 : Y2
 = 100 : Screen  Copy  Logic ( 0
 ) To 1  :  Ink 4  :  Bar  0 ,
 0 To 319 , 9 : Bar 0 , 190 To 319 ,
 199 :  Ink  5 :  Bar  158  ,
 10 To 161 , 189  :  BY = 160  :
 Colour 1, $ F00
40 Bob Update Off : D ( 0 ) = 2 ^
 LVL : D ( 1 ) = - 1 : Ink 2 : Bar 0 ,
 10 To 4 , 189 : Bar 315 , 10 To 319 ,
 189 :  SC1 = 0 : SC2 = - 1 : BX = 1 -
 D ( 0 ) : While SC1 < 5 and SC2 < 5
50 Screen Copy  1  To  Logic ( 0
 ) : Screen 0 : Bob 1 , 8 , Y1 ,
 1 : Bob  2 , 311 , Y2 , 1 : Bob 3 , BX
 , BY , 2 : Bob Update : A = 0 :
 Repeat :  Screen 1 :  P = Point ( Max
 ( 0 , Min ( 319 , BX + D ( 0 ) * ( 1 -
 A ) ) ) , Max ( 0 , Min ( 199 , BY +
 D ( 1 ) * A ) ) ) : Screen 0
60 If Bob Col ( 3 , 1 To 2 ) Then  D
 ( 0 ) = ( - 1 - Col ( 1 ) * 2 ) * ( 2 ^
 LVL ) :  D = - Col ( 2 ) : D ( 1 ) =
 ( ( BY + Y1 * ( D = 0 ) + Y2 * ( D
 = 1 ) ) * 2 ) / ( 4 - LVL ) : A =
 1 : BX = BX + D ( 0 ) : BY = BY + D
 ( 1 ) : Shoot :  Goto 80
70 If P =  2  Then  Boom :
 Wait 10 :  SC1 = SC1 - ( BX > 160
 ) : SC2 = SC2 - ( BX < 160 ) : BX =
 7 : BY = 100 + Rnd ( 60 ) - Rnd ( 60
 ) : D ( 0 ) = 2 ^ LVL : D ( 1 ) = Rnd
 ( 2 ) -Rnd ( 2
 ) : Screen 1 : Paper  0 : Pen
 15 : Locate 3 , 3 : S$ = Str$ ( SC1 )
 + ” :” + Str$ ( SC2 ) + ” ” :
 Print S$ : Screen 0
80 If P = 4 Then Bell : D ( 1 ) = - D (
 1 ) : BY =Max ( 10 , Min ( 189 , BY ) )
90 Inc A : Until A = 2 : DY = 0 : Y0 =
 Y1 : Y1 = Max ( 16 , Min ( 184,
Y Screen ( Y Mouse ) ) ) : DY = Abs (
 Sgn ( Y0 - Y1 ) ) * 2 ^ LVL : Y2 =
 Y2 +Min ( Abs ( BY - Y2 ) , LVL * 3
 ) * Sgn ( BY - Y2 ) : BX = BX + D (
 0 ) : BY = Max ( 2 , Min ( 197 , BY +
 D ( 1 ) + WG * ( Rnd ( 2 ) - Rnd ( 2
 ) ) ) )
100 Wend : Screen 0 : Screen
 Copy Logic ( 0 ) To Physic ( 0 ) :
 Locate 0 , 23 : Centre “Press
 A Key” : Wait Key : Screen Open 0 ,
 320 , 200 , 16 , Lowres : Goto 30
```

☐ Mini Maths 5

Displaying to a fixed number of decimal places.

This is quite simple. Just multiply by 10 for 1Dp, 100 for 2Dp etc (The actual formula is of course 10 to the power of the number of decimal places) Turn it into an integer (to get rid of the other fractional part) and then divide it by 10, 100, 1000 etc. e.g

```
Input “Enter a floating Point number ”
 , F#
A = F# * 100 : rem display to 2
 decimal places
A# = A / 100.0
Print A#
```

The other method is to use strings.

```
Input “Enter a number ” , N#
Input “Enter the number of decimal
 places #”, D
N$ = str$ ( N ) + string$ ( “0” , D )
N$ = left$ ( N$ , Instr ( N$ , ”.” ) +
 D )
Print N$
```

This Newsletter was written by Aaron Fothergill, unless otherwise stated, put together on an Amiga and an ST using Timeworks DTP.

Graphics by Adam Fothergill.

Printed by Creative Printers, 104 Boutport Street, Barnstaple, N.Devon. Telephone (0271) 77655